# Improving Continuous-time Conflict Based Search[*]

**Anton Andreychuk,**[1, 2] **Konstantin Yakovlev,** [2, 3] **Eli Boyarski,** [4] **Roni Stern** [4, 5]
[1]Peoples' Friendship University of Russia (RUDN University)
[2]Federal Research Center for Computer Science and Control of Russian Academy of Sciences
[3]HSE University [4]Ben-Gurion University of the Negev
[5]Palo Alto Research Center
andreychuk@mail.com, yakovlev@isa.ru, eli@boyar.ski, sternron@post.bgu.ac.il

**Abstract**

Multi-Agent Pathfinding (MAPF) is the problem of finding paths for $n$ agents in a graph such that each agent reaches its goal vertex and the agents do not collide with each other while moving along these paths. While different problem statements of MAPF exist, we are focused on MAPF$_R$ (Walker, Sturtevant, and Felner 2018), in which actions' durations can be non-uniform, agents have geometric shapes, and time is continuous. Continuous-time conflict-based search (CCBS) (Andreychuk et al. 2019) is a recently proposed algorithm for finding optimal solutions to MAPF$_R$ problems. In this work, we propose several improvements to CCBS based on known improvements to the Conflict-based search (CBS) algorithm (Sharon et al. 2015) for classical MAPF, namely Disjoint Splitting (DS), Prioritizing Conflicts (PC), and high-level heuristics. We evaluate the impact of these improvements experimentally on both roadmaps and grids. Our results show that CCBS with these improvements is able to solve significantly more problems.

## Continuous-Time Conflict Based Search

CCBS is a modification of CBS algorithm that can solve MAPF$_R$ problems. To consider continuous time, CCBS reasons over the *time intervals*, detects conflicts between *timed actions*, and resolves conflicts by imposing constraints that specify the time intervals in which the conflicting timed actions can be moved to avoid the conflict. Formally, a CCBS conflict is a tuple $(a_i, t_i, a_j, t_j)$, specifying that the timed action $(a_i, t_i)$ of agent $i$ has a conflict with the timed action $(a_j, t_j)$ of agent $j$. A CCBS constraint is a tuple $(i, a_i, [t_i, t_i^u))$ specifying that agent $i$ cannot perform action $a_i$ in the time interval $[t_i, t_i^u)$. The low-level planner of CCBS is an adaptation of the Safe interval path planning (SIPP) algorithm (Phillips and Likhachev 2011).

## Disjoint Splitting for CCBS

The first technique we migrate from CBS to CCBS is Disjoint Splitting (DS) (Li et al. 2019). CBS with DS (CBS-DS) considers *positive* and *negative* constraints. A *negative* constraint $\overline{(i, x, k)}$ states that agent $i$ *must not* be at $x$ at time step $k$. A *positive* constraint $(i, x, k)$ means that agent $i$ *must* be at $x$ at time step $k$. When splitting a Constraint Tree (CT) node $N$ over a CBS conflict $(i, j, x, k)$, CBS-DS chooses one of the conflicting agents, say $i$, and generates two child nodes, one with the negative constraint $\overline{(i, x, k)}$ and the other with the positive constraint $(i, x, k)$.

A CCBS constraint $(i, a_i, [t_i, t_i^u))$ can be stated formally as follows: $\forall t \in [t_i, t_i^u) : (a_i, t)$ is *not* in a plan for agent $i$. This is a negative constraint from a DS perspective. The corresponding positive constraint is therefore the inverse: $\exists t \in [t_i, t_i^u) : (a_i, t)$ is in a plan for agent $i$. This mean that agent $i$ must perform $a_i$ at some moment of time from the given interval. Thus a positive constraint in CCBS is an *action landmark*, i.e., the action that must be performed in any solution. Consider now an action landmark $l = (i, \text{move}(A, B), [t, t^u))$. In CCBS+DS there is an additional challenge for the low-level search: there may be infinite plans that satisfy $l$, i.e., reach $A$ within $[t, t^u)$. Finding only the least-cost plan might lead to incompleteness. To guarantee completeness and optimality we need to find the lowest-cost plan of reaching $A$ *for every safe interval* of $A$ that overlaps with $[t, t^u)$. To solve this issue, we introduce a generalized version of SIPP such that: (1) it accepts a set of goal states, one per safe interval of $A$ that overlaps with $[t, t^u)$, and (2) it outputs a set of plans, one per goal state. To each of these plans, we concatenate the action landmark itself move$(A, B)$. These plans may end in different safe intervals in $B$, which then become distinct start states when searching for a plan to get from $B$ to the next landmark.

## Prioritizing Conflicts

Prioritizing Conflicts (PC) (Boyarski et al. 2015) is the second CBS enhancement we migrate to CCBS. PC is a heuristic for choosing which conflict to resolve when expanding a CT node. PC prioritizes conflicts by classifying each conflict as either *cardinal*, *semi-cardinal*, *non-cardinal*. However, In MAPF$_R$, most conflicts are cardinal, i.e., the agents involved in that conflicts are not able to find the paths that respect the corresponding constraints and are of the same cost as before. To this end, we propose a generalized version of PC that introduces a finer-grained prioritization of conflicts, by

[*]This is the extended abstract for the same paper presented at AAAI'21.

introducing the notion of *cost impact*. For a CT node $N$ with a CCBS conflict $Con = (a_i, t_i, a_j, t_j)$, let $N_i$ and $N_j$ be the CCBS nodes obtained by splitting over this conflict, and let $\delta_i$ be the difference between the cost of $N$ and $N_i$. We define the *cost impact* of the conflict $Con$, denoted $\Delta(Con)$, as $\min(\delta_i, \delta_j)$ Our adaptation of PC to CCBS chooses to split a CT node on the conflict with the largest cost impact.

## Heuristics for High-Level Search

Drawing from (Felner et al. 2018) we suggest two admissible heuristics for CCBS. The first heuristic is based on solving the linear programming problem (LPP) with $n$ non-negative variables $x_1, \dots x_n$, one for each agent. Each conflict $Con_{i,j}$ between agents $i$ and $j$ introduces the LPP constraint $x_i + x_j \geq \Delta(Con_{i,j})$. The objective to be minimized is $\sum_{i=1}^{n} x_i$. The second heuristic follows the approach suggested in (Felner et al. 2018). We, first, sort the conflicts in descending order of their cost impact. Then, conflicts are picked one by one in this order. After a conflict is picked, we remove from the conflict list all conflicts that involve any of the agents in this conflict. This continues until all the conflicts are either picked or removed. The resultant heuristic is the sum of the cost impacts of the chosen conflicts. We observed experimentally that the practical difference between the described heuristics is negligible. In the experiments described below we used the second heuristic.

## Empirical Evaluation

The experiments were conducted on $2^3$- and $2^5$-connected grids from the MovingAI MAPF benchmark (Stern et al. 2019): a 16x16 empty grid, a warehouse-like grid (warehouse-10-20-10-2-2), and a large game grid (den520d). We also generated 3 roadmaps from den520d grid with different density of nodes and edges: sparse, dense and super-dense.[1] All the agents were assumed to be disk-shaped with $\sqrt{2}/4$ radius. The number of agents varied from 2 to 100. The time limit for each run was 30 seconds.

The results of all experiments are shown in Table 1. We can see that in almost all cases the best results were obtained by CCBS with all our enhancements (last column). In certain setups, e.g., on sparse roadmap, the increase in the number of solved instances is two-fold. Comparing the results on grids with different neighborhood sizes, one can notice the differences in benefits of PC and DS: increasing the branching factor makes PC less effective and DS more effective. This is explained by the fact that higher branching factor means stronger pruning by positive constraints. The similar trend is observed on the roadmaps. The impact from adding the heuristic is not pronounced in terms of the number of solved instances. However, using heuristic leads to the reduction of the CT-nodes expansions (up to 15-25%).

| | Basic | PC | DS | DS+PC | All |
|---|---|---|---|---|---|
| 16x16_k3 | 392 | 455 | 506 | 587 | **595** |
| warehouse_k3 | 771 | 1135 | 901 | **1163** | **1163** |
| den520d_k3 | 667 | 766 | 729 | 809 | **810** |
| 16x16_k5 | 239 | 254 | 366 | 406 | **410** |
| warehouse_k5 | 633 | 821 | 846 | 917 | **925** |
| den520d_k5 | 296 | 301 | **472** | 451 | 451 |
| sparse | 239 | 389 | 329 | 468 | **476** |
| dense | 344 | 392 | 494 | 507 | **520** |
| super-dense | 211 | 206 | **367** | 309 | 309 |

Table 1: Number of problems solved within the time limit.

## Conclusions and Future Work

In this work, we have proposed three improvements for the CCBS algorithm. The first one, called DS, changes how CT nodes are expanded by introducing positive and negative constraints. The second one, called PC, prioritizes the conflicts to resolve by computing the cost of the solution that resolves them. The last improvement is two admissible heuristics for the high-level search. In a comprehensive experimental evaluation, we observed that using these improvements, CCBS can scale to solve much more problems than the basic CCBS, solving in some cases almost twice as many agents. Future work may consider adapting additional improvements from CBS to CCBS, such as bipartite reduction (Walker, Sturtevant, and Felner 2020) and symmetry-breaking techniques (Li et al. 2020).

[1]Our implementation and all the raw results are available at: `github.com/PathPlanning/Continuous-CBS`.

## References

Andreychuk, A.; Yakovlev, K.; Atzmon, D.; and Stern, R. 2019. Multi-Agent Pathfinding with Continuous Time. In *IJCAI 2019*, 39–45.

Boyarski, E.; Felner, A.; Stern, R.; Sharon, G.; Betzalel, O.; Tolpin, D.; and Shimony, E. 2015. ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding. In *IJCAI 2015*, 740–746.

Felner, A.; Li, J.; Boyarski, E.; Ma, H.; Cohen, L.; Kumar, T. S.; and Koenig, S. 2018. Adding Heuristics to Conflict-Based Search for Multi-Agent Path Finding. In *ICAPS 2019*, 83–87.

Li, J.; Gange, G.; Harabor, D.; Stuckey, P. J.; Ma, H.; and Koenig, S. 2020. New techniques for pairwise symmetry breaking in multi-agent path finding. In *ICAPS 2020*, 193–201.

Li, J.; Harabor, D.; Stuckey, P. J.; Felner, A.; Ma, H.; and Koenig, S. 2019. Disjoint splitting for multi-agent path finding with conflict-based search. In *ICAPS 2019*, 279–283.

Phillips, M.; and Likhachev, M. 2011. SIPP: Safe interval path planning for dynamic environments. In *ICRA 2011*, 5628–5635.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant., N. R. 2015. Conflict-based search for optimal multiagent path finding. *Artificial Intelligence Journal* 218: 40–66.

Stern, R.; Sturtevant, N. R.; Felner, A.; Koenig, S.; Ma, H.; Walker, T. T.; Li, J.; Atzmon, D.; Cohen, L.; Kumar, T. S.; et al. 2019. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *SoCS 2019*, 151–158.

Walker, T. T.; Sturtevant, N. R.; and Felner, A. 2018. Extended Increasing Cost Tree Search for Non-Unit Cost Domains. In *IJCAI 2018*, 534–540.

Walker, T. T.; Sturtevant, N. R.; and Felner, A. 2020. Generalized and Sub-Optimal Bipartite Constraints for Conflict-Based Search. In *AAAI 2020*, 7277–7284.